Chapter 3: Strings, Vectors and Arrays

Namespace

- Mechanism for putting names defined by a library into a single logical place.
- Namespaces help avoid name clashes (抵觸). The names defined by the C++ library are in the namespace std.

```
std::cout
```

- A using declaration allows us to access a name from a namespace without the cumbersome prefix namespace_name:: (e.g., std::)
- See note

Headers **Should Not** Include using Declaration

- Inside header files, we should *always* use the fully qualified library names, that is, DO NOT use using declaration. (why?)

- **A:** Avoid unexpected name conflicts. If we place a using declaration within a header, it is equivalent to placing the same using declaration in every program that includes the header *whether that program wants the using declaration or not*.

string type

- The string type supports variable-length character strings.
- The library takes care of managing the memory and provides various useful operations.

string s1;	**Default constructor; s1 is the empty string**
string s2(s1);	**Initialize s2 as a copy of s1**
string s3("value");	**Initialize s3 as a copy of the string literal**
string s4(n, 'c');	**Initialize s4 with n copies of the character 'c'**

string I/O

```
string s;
cin >> s;
```

- Reads and discards any leading whitespace (e.g., spaces, newlines, tabs)
- It then reads characters until the next whitespace character is encountered.

```
string line;
getline(cin, line);
```

- Reads the next line of input stream and store what it reads, not including the newline.
- See note

Operations on strings

- Table 3.2 (see note).
- The empty operation
- The string size operation and its machine independent return type (string::size_type, see note)
- The string relational operations
- Assignment for strings
- Adding two strings
- Adding character string literal and strings
- Subscript [ ] (out-of-range problem!)

Exercise, see note

Dealing with characters in a string

- See Table 3.3 cctype function (see note).

```
for (string::size_type index = 0; index != s.size();
     ++index)
    if (ispunct(s[index])) ++punct_cnt;

for (auto index = 0; index != s.size(); ++index)
    if (ispunct(s[index])) ++punct_cnt;


for (auto e: s)
    if (ispunct(e)) ++punct_cnt;
```

Exercise, see note

vector type

- A vector is a collection of objects of a single type, each of which has an associated integer index.
- A vector is a **class template**. To declare objects of a type generated from vector, we must supply what type of objects the vector will contain. We specify the type by putting it between a pair of angle brackets following the template's name:

```
vector<int> ivec;
vector<Sales_item> salesVec;
vector<vector<int> > matInt;
```

Defining and initializing vector

- Table 3.4 (see note).

```
vector<int> ivec(10, -1);
vector<string> svec(10, “hi”);

 vector<int> ivec(10);
 vector<int> ivec(10, 1);
 vector<int> ivec{10, 1};

 vector<string> svec(10);
 vector<Sales_item> salesVec(10);
```

Operations on vector

- See Table 3.5 (pp. 93).
- The empty operation
- The vector size operation and its machine independent return type (what is it?)
- Adding elements to a vector

```
vector<string> svec(10, “hi”);
svec.push_back(“there”);
```

  - See note.
- Subscript [ ] (out-of-range problem!)

Until Next Time

- Lab starts at 6:00 pm on Thurs.
- HW1 will be due at 0900 pm on 09.24 (late submission policy)
- [Reading] Chapter 3 cont., Chapter 4-7.